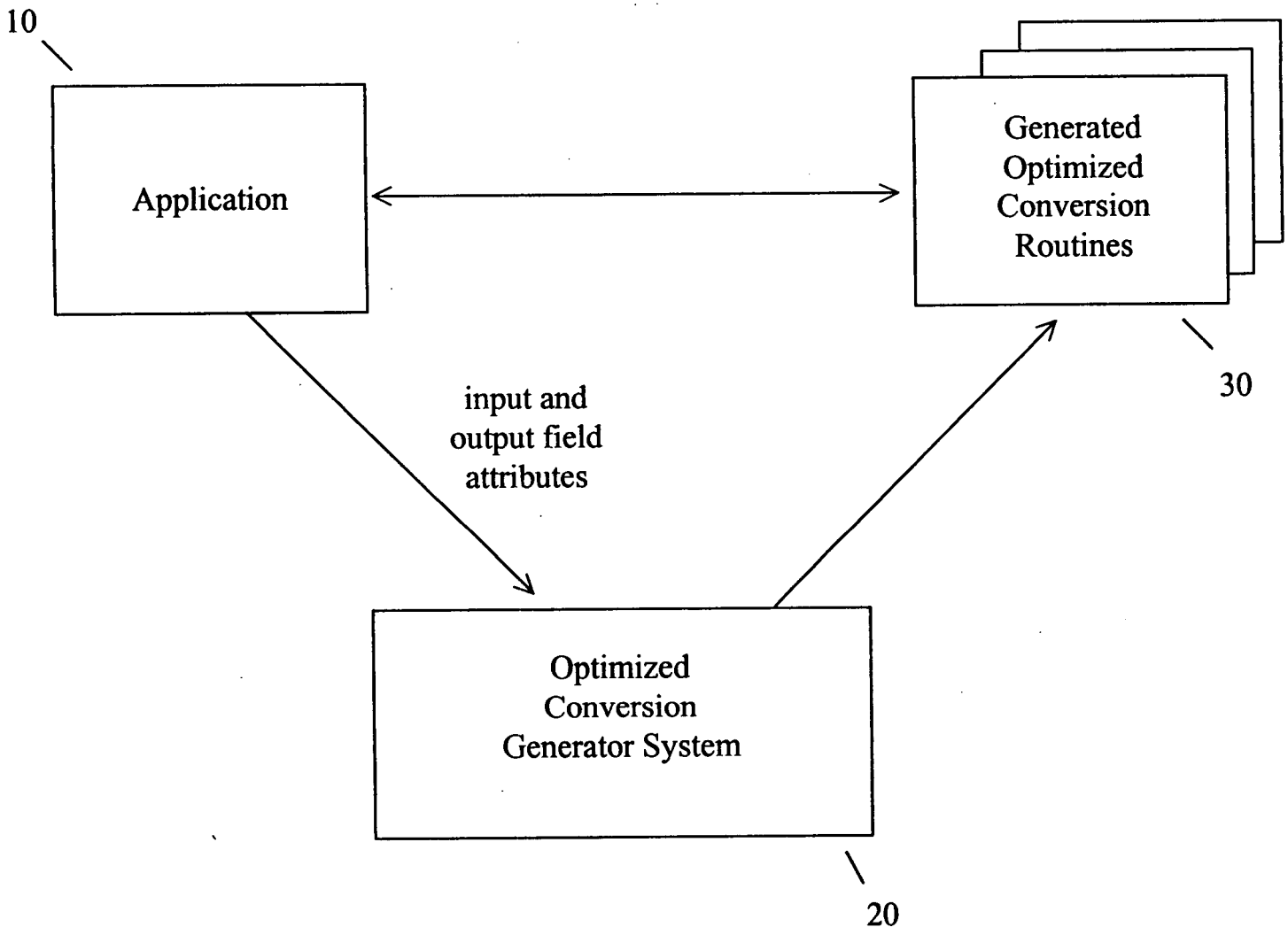

Fig. 1

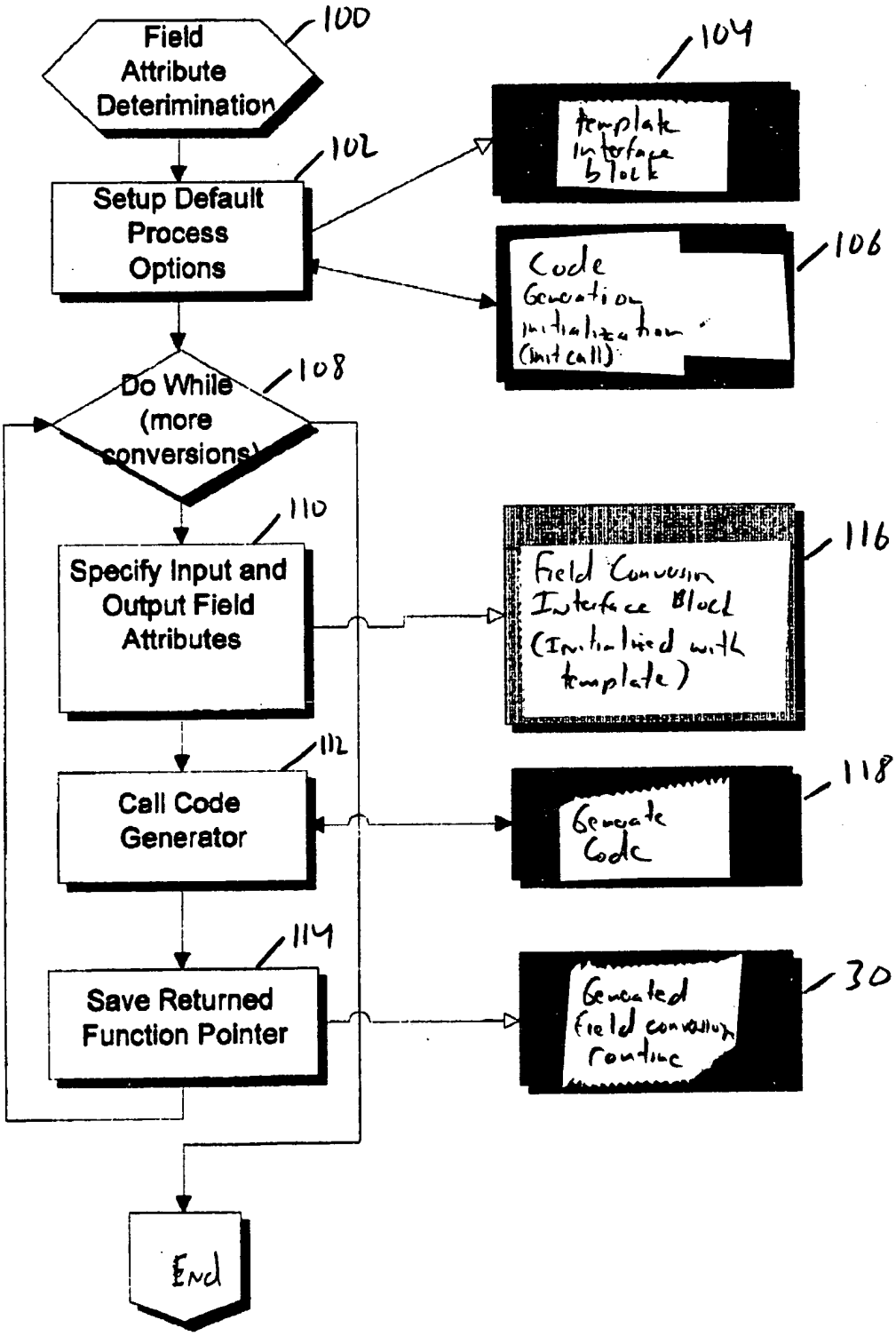
Field Attribute Deterimination
100
Setup Default Process Options
102
Do While (more conversions)
108
Specify Input and Output Field Attributes
110
Call Code Generator
112
Save Returned Function Pointer
114
End
template interface block
104
Code Generation initialization (init call)
106
Field Conversion Interface Block (Initialized with template)
116
Generate Code
118
Generated field conversion routine
30

Fig. 2

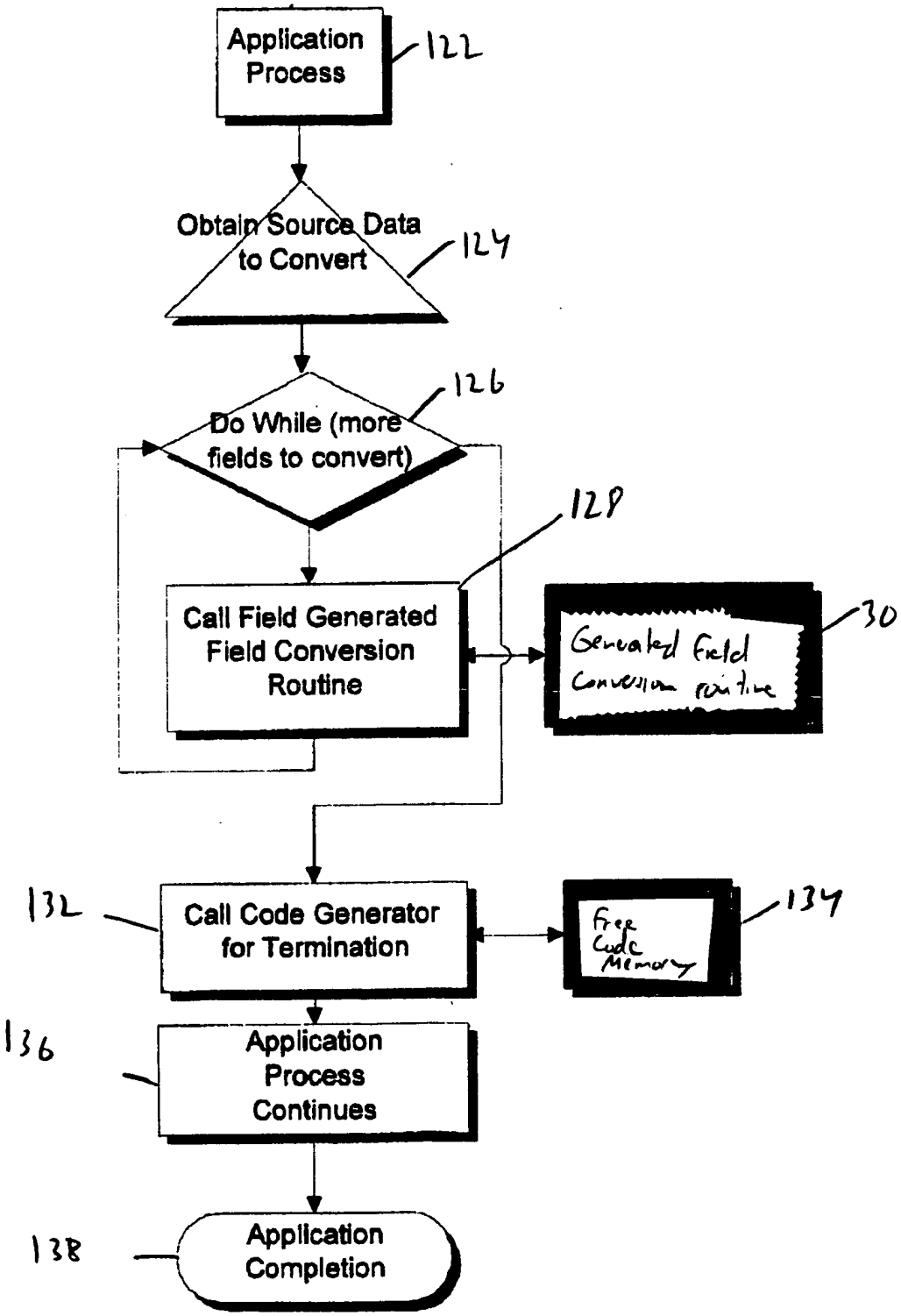
Application Process
122
Obtain Source Data to Convert
124
Do While (more fields to convert)
126
Call Field Generated Field Conversion Routine
128
Generated Field Conversion routine
130
Call Code Generator for Termination
132
Free Code Memory
134
Application Process Continues
136
Application Completion
138

Fig. 3

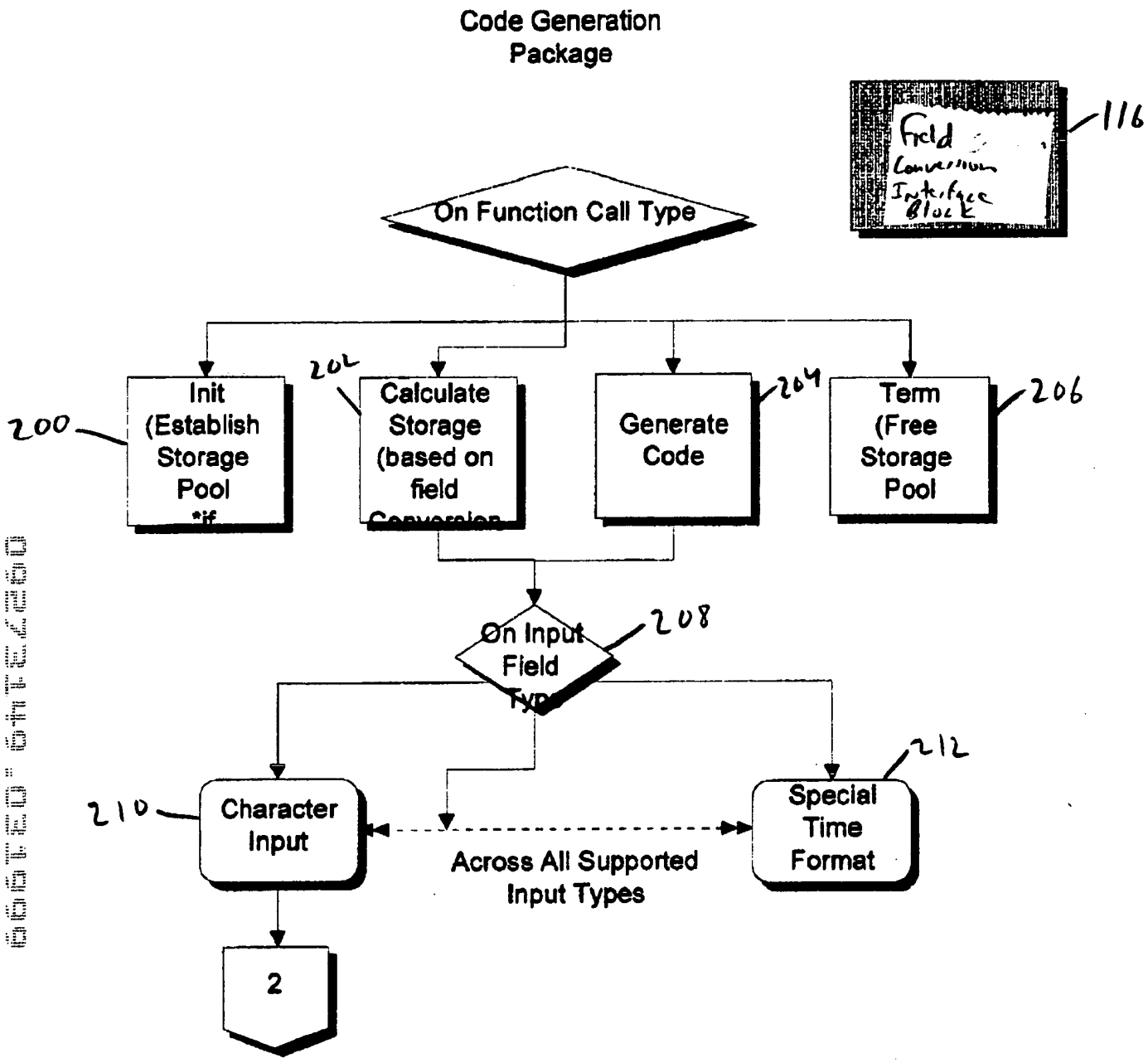
Code Generation Package
Field Conversion Interface Block
116
On Function Call Type
200
Init (Establish Storage Pool
202
Calculate Storage (based on field
Generate Code
204
Term (Free Storage Pool
206
On Input Field
208
210
Character Input
212
Special Time Format
Across All Supported Input Types
2

Fig. 4a

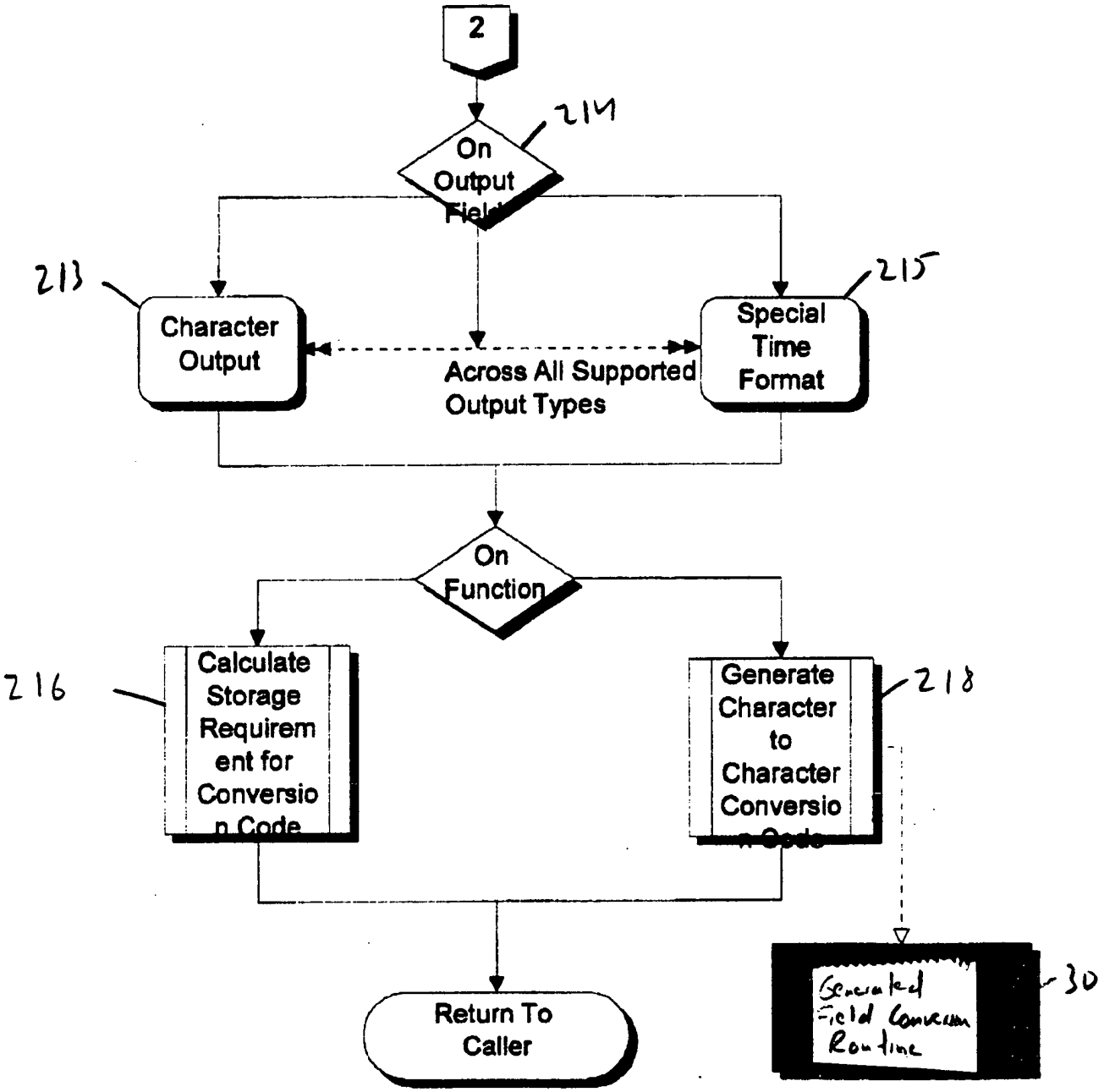
2
214
On Output Field
213
Character Output
215
Special Time Format
Across All Supported Output Types
On Function
216
Calculate Storage Requirement for Conversion Code
218
Generate Character to Character Conversion Code
Generated Field Conversion Routine
30
Return To Caller

Fig. 4b

```
R5    =     Current Instruction Offset within application buffer
R6    =     Current Instruction Address within application buffer
R7    =     Work Register - used for calculating offsets, etc
R12   =     Base register of code generator and template code

SLR         R5,R5                    clear offset
L           R6,$BCB_BCODE_@          get address of user buffer

* if linkage required call standard linkage builder
IF (TM,$BCB_PFLAG1,$BCB_LINKAGE,O)
   SETF    LINKAGE
   IF (CLI,$BCB_LINKAGE_TYPE,EQ,C'N')
      RESETF  LINKAGE
   COND ELSE
* call standard linkage builder
      #BAS    14,=A(BURST_ENTRY_LINKAGE)
   ENDIF
ELSE
   RESETF  LINKAGE
ENDIF
****
STDRETURN          -     RETURN TO APPLICATION
* $BCB_BCODE_@ WILL POINT TO BUILT CODE
****

*
* Routine to build standard entry linkage
*
BURST_ENTRY_LINKAGE CSMSUBI BASE=R10,WORKREG=R3
*
* Move Template code into user buffer
       MVC   0(STD_ENTL_010_L,R6),STD_ENTL_010
*
* Modify  " LA     R14,0(0)" instruction
* Get Offset to Savearea using equate STD_ENTL_010_SA_A
* Set base register for instruction to R12
* Set D(X,B) of instruction (R7 contains constructed D(X,B))
       LA    R7,STD_ENTL_010_SA_A(,R5)
       O     R7,=X'0000C000'
       STH   R7,STD_ENTL_010_SA_T(,R6)
*
```

Fig. 5a

```
* Modify  " B      0(R12)" instruction
* Get offset of branch target using equate STD_ENTL_010_B_A_T
* Set D(X,B) of instruction (R7 contains constructed D(X,B))
* ** Note X (index register) has been set by assembler as R12
*    STH does not change the instruction's index register
       LA    R7,STD_ENTL_010_B_A_T(,R5) CALC OFFSET FOR BRANCH TARGET
       STH   R7,STD_ENTL_010_B_A(,R6)   SET BRANCH D(X,B)
*
* Increment Next Instruction Offset (in R5) by length of code
* Increment Next Instruction Address (in R6) by length of code
       LA    R5,STD_ENTL_010_L(,R5)
       LA    R6,STD_ENTL_010_L(,R6)
*
* Return to caller
* Code has been built and the Instruction Offset and Address registers
* have been updated for next instruction construction

           CSMSUBO
*- STANDARD ENTRY LINKAGE ----------------------------------------
*
*-----------------------------------------------------------------
STD_ENTL_010 DS  0S
             STM    R14,R12,12(R13)
STD_ENTL_010_SA_T EQU  *-STD_ENTL_010+2
             LA     R14,0(0)                  BURSTED SAVEAREA+0
             ST     R13,4(,R14)
             ST     R14,8(,R13)
             LR     R13,R14                   R13 = BURSTED SAVEAREA
             LR     R12,R15                   SET BURSTED BASE REG
STD_ENTL_010_B_A  EQU  *-STD_ENTL_010+2
             B      0(R12)                    WS_BRANCH
STD_ENTL_010_SA_A EQU  *-STD_ENTL_010
             DC     18F'0'
STD_ENTL_010_B_A_T EQU  *-STD_ENTL_010
STD_ENTL_010_L   EQU   *-STD_ENTL_010
*-----------------------------------------------------------------
```

Fig. 5b

```
* Call made by API passing API $BURSTCB control block
* Control block contains field attributes and conversion
* options

* Reset processing flags
*  NO_BUILD -> doing conversion routine storage calculation
*  CALLED_ROUTINE -> creating a called routine
* Check for API block -> if not there abend with dump
* Copy passed API block to working storage (IN_BCB)

MAIN_0000 DS      0S
*
          RESETF  NO_BUILD
          RESETF  CALLED_ROUTINE
*
          LTR     R1,R1
          BNZ     MAIN_0005
*
          ABEND   001,DUMP
*
MAIN_0005 DS      0S
          MVC     IN_BCB($BCB_LENGTH),0(R1)
*
          LA      R9,IN_BCB              R9 = ADDRESS OF $BURSTCB
          USING   $BURSTCB,R9
*
* If calculate storage requested SET NO_BUILD
          IF (CLC,$BCB_FUNC,EQ,=Y($BCB_CALC_STORAGE))
             SETF    NO_BUILD
          ENDIF

* INITIALIZE WORKING STORAGE
* If actually BUILDING code (not NO_BUILD)
*  1. Obtain offset from beginning of BASE REGISTER
*    for code.  If callable routine this has been set to 0.
*    otherwise this we are building inline code within the application's
*    user managed buffer and the offset will set to current instruction offset
*    within the buffer.
*     2. Obtain address of passed code buffer
*     3. Calculate current instruction address based on offset into buffer
```

Fig. 6a

```
MAIN_STRT DS      0S
          IF (¬NO_BUILD)
             LH      R5,$BCB_BCODE_OFFSET
             L       R6,$BCB_BCODE_@
             LA      R6,0(R5,R6)
          ELSE
             SLR     R5,R5                    CLEAR FOR ACCUM
             SLR     R6,R6                    CLEAR FOR ACCUM
          ENDIF
*
* INITIALIZE WORK FIELDS FOR ANY COLUMN CONVERSION
* 1. Obtain input field's addressing register
* 2. Build RX type assembler instruction D(X,B) with offset 0
* 3. Obtain output field's addressing register
* 4. Build RX type assembler instruction D(X,B) with offset 0
*    set template for output D(X,B)
* 5. Obtain input and output lengths
* 6. Set Current working D(X,B) templates
          SLR     R7,R7
          ICM     R7,B'0001',$BCB_IREG
          SLL     R7,4                             SHIFT NIBBLE
          STC     R7,WB_INIT_SOURCE_DB
          ICM     R7,B'0001',$BCB_OREG
          SLL     R7,4                             SHIFT NIBBLE
          STC     R7,WB_INIT_TARGET_DB
          MVC     WB_TOT_INPUT_LEN,$BCB_ILEN
          MVC     WB_TOT_OUTPUT_LEN,$BCB_OLEN
          MVC     WB_SOURCE_DB,WB_INIT_SOURCE_DB   RESET DB
          MVC     WB_TARGET_DB,WB_INIT_TARGET_DB   RESET DB
*
* CHECK FOR LINKAGE REQUIREMENTS
* IF LINKAGE = E (BASIC ENTRY - SAVE/RESTORE R14) THEN
*   BURST_WORK_BRANCH WILL SAVE R14 AND SET RESTORE_R14
*   BURST_EXIT_LINKAGE RESTORES R14 AND BASR R14
* ENDIF
          RESETF  RESTORE_R14
          IF (TM,$BCB_PFLAG1,$BCB_LINKAGE,O)
             SETF    LINKAGE
             IF (CLI,$BCB_LINKAGE_TYPE,EQ,C'N')
                RESETF  LINKAGE
             COND ELSE
                #BAS    14,=A(BURST_ENTRY_LINKAGE)
             ENDIF
          ELSE
             RESETF  LINKAGE
          ENDIF
```

Fig. 6b

```
* CALL INPUT TYPE PROCESSING ROUTINE
* 1. Get address of input field type table
*    This table contains an index of supported input types
*    with their associated code generation routines
* 2. Call code generation routine for Input field type
*    In this case INPUT FIELD TYPE IS CHARACTER
*    INPUT FIELD TYPE CHARACTER calls routine named CHARACTER
**** Further down subroutine CHARACTER is shown
          L       R14,=A(TYPE_TABLE)
          LH      R15,$BCB_ITYPE
          LA      R15,0(R14,R15)
          L       R15,0(,R15)
          BASR    R14,R15

* Subroutine has built conversion code for INPUT TYPE CHARACTER and OUTPUT TYPE CHARACTER
* Check for other process options such as: accumulate a source addressing register,
* accumulate a target addressing register, or accumulate alternate register.
* alternate register usually is a total output length accumulator used by the calling
* application to keep track of an aggregate of all output field lengths
* 1. IF source addressing register accumulate requested build code to accumulate
* 2. IF target addressing register accumulate requested build code to accumulate
* 3. IF length register accumulate requested build code to accumulate
* 4. IF exit linkage requested build exit linkage
* 5. RETURN TO API CALLER with generated conversion routine
MAIN_0200 DS      0S
          IF (TM,$BCB_PFLAG1,$BCB_SRC_ACUM,O)
             LH     R0,WB_SOURCE_ACCUM_INDEX
             IC     R1,$BCB_SRC_ACUM_REG
             LH     R7,WB_TOT_INPUT_LEN
             #BAS   14,=A(FIXED_ACCUM)
          ENDIF
*
          IF (TM,$BCB_PFLAG1,$BCB_TRG_ACUM,O)
             LH     R0,WB_TARGET_ACCUM_INDEX
             IC     R1,$BCB_TRG_ACUM_REG
             LH     R7,WB_TOT_OUTPUT_LEN
             #BAS   14,=A(FIXED_ACCUM)
          ENDIF
*
          IF (TM,$BCB_PFLAG1,$BCB_TRG_L_ACUM,O)
             LH     R0,WB_TARGET_ACCUM_INDEX
             IC     R1,$BCB_TLN_ACUM_REG
             LH     R7,WB_TOT_OUTPUT_LEN
             #BAS   14,=A(FIXED_ACCUM)
          ENDIF
*
* BURST EXIT LINKAGE
          IF (LINKAGE)
             SETF    CLEAR_R15
             #BAS    14,=A(BURST_EXIT_LINKAGE)
          ENDIF
          RETURN to CALLER
```

Fig. 6c

```
*-------------------------------------------------------------------*
* Character Input Field Type Conversion Routine                     *
* Abstract:                                                         *
*     This routine is called to either build Character Input        *
*     Fields to all supported Output Field Types, or to calculate   *
*     storage requirements for generated conversion routines for    *
*     Input field type Character                                    *
*                                                                   *
* CHARACTER field type constraints                                  *
*    These field types will be of fixed length                      *
*    Maximum length is 254 8bit bytes                               *
*    They may be proceeded with a null field indicator of length    *
*       1 byte that will contain values of x'00' for non-null fields*
*       and x'ff' for nulled fields.  Nulled fields will not be     *
*       converted accept to indicate on output that field was null  *
*    There values are of EBCDIC CCSID (character code set) unless   *
*    a CCSID is specified through the API.                          *
*                                                                   *
*-------------------------------------------------------------------*


CHARACTER CSMSUBI BASE=R10,WORKREG=R3
* Use branch table generated by API to branch on output type (BTYPE=O)
* Example is demonstrating character to character conversion
* Branch will be taken to CHAR_CHAR_0000
     L        R15,=A(RET_RC_32)
     $BURST   BTABLE,                                              X
              BREG=1,                                              X
              BTYPE=O,                                             X
              UNSUPPORTED=0(,R15),                                 X
              CHAR=CHAR_CHAR_0000,                                 X
              LVARC=CHAR_VARC_0000,                                X
              VARC=CHAR_VARC_0000

*-@PSEUDO-CODE@-----------------------------------------------------
*             CHARACTER TO CHARACTER CONVERSION
*
* -DETERMINE WORKING STORAGE
*   Some conversions require the generation of local working storage
*   Working storage is generated according to specific conversion options and
*   specific input and output field attributes to avoid generating more storage
*   than needed.
*
*   IF CONVERTING CCSID'S (character code sets) THEN
*     IF using a character translation table (uses TR instruction)
*       Build BRANCH over working storage
*       Build FULL WORD to hold Address character translation table
*       UPDATE Previously built Branch instruction to branch to current offset
*          (offset is next halfword aligned byte where next instruction is to be built)
*      ENDIF
*   ENDIF
*
```

Fig. 6d

```
*   IF INPUT LENGTH is GREATER than OUTPUT LENGTH
*      current implementation allows for truncation of trailing spaces
*      If input field being converted by generated code contains non-spaces
*         that won't fit into output field of lesser length then conversion
*         error 4 routine will be called to return a value of 4 in R15
*
*      1. Build BRANCH over working storage
*      2. Build a buffer full of spaces to be used in INPUT field compare
*      3. Build Conversion error routine to return error #4
*      4. UPDATE Previously built Branch instruction to branch to current offset
*           (offset is next halfword aligned byte where next instruction is to be built)
*    ENDIF
* - DETERMINE WORKING STORAGE
*
*-@PSEUDO-CODE@-------------------------------------------------------CHAR_CHAR_0000 DS 0S
*
* BURST WORKAREA IF CONVERSION ERROR OR CONVERT CCSID
          TM     $BCB_PFLAG2,$BCB_CCSID_CNV
          BNZ    CHAR_CHAR_0020
          CLC    $BCB_ILEN,$BCB_OLEN
          BNH    CHAR_CHAR_0040
*
CHAR_CHAR_0020 DS 0S
          #BAS   14,=A(BURST_WORK_BRANCH)
*
          IF (TM,$BCB_PFLAG2,$BCB_CCSID_CNV,NZ)
            IF (TM,$BCB_PFLAG2,$BCB_CCSID_CNV_ATOE,O)
               #BAS   14,=A(BURST_BWK_TO_E_XLATE_@)
            ELSE
               #BAS   14,=A(BURST_BWK_TO_O_XLATE_@)
            ENDIF
            #BAS   14,=A(BURST_BWK_FULL)
            STH    R7,WB_SAVE_R2_OFFSET
          ENDIF
*
* IF ILEN > OLEN THEN NEED FOLLOWING WORK FIELDS
*   BURST BUFFER255 - SPACES
*    BURST #@ERROR4 CALL
* ENDIF
          IF (CLC,$BCB_ILEN,GT,$BCB_OLEN)
             #BAS   14,=A(BURST_BWK_BUFFER255)
*
*
             LA     R1,4                          #@ERROR4
             #BAS   14,=A(BUILD_CNVERR)
          ENDIF
*
          #BAS   14,=A(UPDATE_WORK_BRANCH)
*
*
```

Fig. 6e

```
* IF OUTPUT NULLABLE THEN
*   BURST MOVEMENT OF NULL INDICATOR
*   R1 = X'00' FOR MVI Instruction Builder
*   WB_TARGET_DB (current target D(B)) USED FOR INDICATOR LOCATION
*   Build MVI OF NULL INDICATOR (MVI_0000)
*   UPDATE Current TARGET D(B) TO ALLOW DATA TO SKIP NULL INDICATOR
*   ADD 1 TO TOT OUTPUT LENGTH (FOR NULL INDC) (this allows for accumulation requests)
* ENDIF
CHAR_CHAR_0040 DS 0S
          IF (TM,$BCB_OFLAG1,$BCB_ONULL,O)
             SLR    R1,R1                          CLEAR SOURCE BYTE
             #BAS   14,=A(MVI_0000)                BURST MVI NULL INDC
*
             LH     R1,WB_TARGET_DB                UPDATE TARGET DB
             LA     R1,1(,R1)
             STH    R1,WB_TARGET_DB
*
             LH     R1,WB_TOT_OUTPUT_LEN           UPDATE OUTPUT LEN
             LA     R1,1(,R1)
             STH    R1,WB_TOT_OUTPUT_LEN
          ENDIF
*
*   IF input length < then output length
*     call routine to build code to pad output field with spaces
*   ELSE
*     IF input length = Output length
*       Call routine to build an MVC instruction
*            This routine uses current source and target D(B)'s
*            and the output length to construct the instruction
*     ELSE
*        input length > output length
*       Call routine to build an MVC instruction
*            This routine call will use the input length (since it shorter)
*            (source and target D(B)'s will be used
*        Build Code to check for truncation of only spaces
*     ENDIF
*   ENDIF
          LH     R1,$BCB_ILEN                    GET INPUT LEN
          LH     R2,$BCB_OLEN                    GET OUTPUT LEN
*
          CR     R1,R2                           CHECK LENGTHS
          BE     CHAR_CHAR_0050                     EQUAL
          BH     CHAR_CHAR_0100                     I > O ->
*
* INPUT LENGTH LESS THAN OUTPUT -> NEED TO PAD
* Build Character padding code
          #BAS   14,=A(SSP_0000)
*
* Build code TO MOVE CHARACTER FIELD TO CHARACTER FIELD
CHAR_CHAR_0050 DS     0S
          #BAS   14,=A(MVC_0000)                 BURST MVC INSTRUCTION
          B      CHAR_CHAR_0200
*
```

Fig. 6f

```
* INPUT field is too large to fit
* Build code TO MOVE CHARACTER FIELD TO CHARACTER FIELD using input field's length
CHAR_CHAR_0100 DS     0S
          LR     R1,R2
          #BAS   14,=A(MVC_0000)                  BURST MVC INSTRUCTION
*
* MOVE CHECK FOR SPACES
* IF TRUNCATED DATA NOT SPACES THEN #@ERROR4

          IF (¬NO_BUILD)
*
          MVC    0(CHAR_CHAR_010_L,R6),CHAR_CHAR_010
*
* SET LENGTH OF COMPARE
          LH     R7,$BCB_ILEN
          SR     R7,R1
          BCTR   R7,0
          STC    R7,CHAR_CHAR_010_OLEN_A(,R6)
*
* SET SOURCE DB TO SOURCE + OLEN-1
          LH     R7,WB_SOURCE_DB
          LA     R7,0(R1,R7)
          BCTR   R7,0
          STH    R7,CHAR_CHAR_010_SDBN_A(,R6)
*
* UPDATE BUFFER OFFSET
          LH     R7,WB_BUFFER255_OFFSET
          O      R7,=X'0000C000'
          STH    R7,CHAR_CHAR_010_B255_A(,R6)
*
* UPDATE #@ERROR4 BRANCH
          LH     R7,WB_CNVERR4_OFFSET
          STH    R7,CHAR_CHAR_010_BERR_A(,R6)
*
          ENDIF                (NO_BUILD)
*
          LA     R5,CHAR_CHAR_010_L(,R5)
          LA     R6,CHAR_CHAR_010_L(,R6)
*
```

Fig. 6g

```
* CHECK FOR TRANSLATION of CCSID's
* If translation requested call translation routine generator
*   *** note translation routine will perform accumulation
*       operation if API requested it.  If accumulation is performed
*       by the routine the IN_BCB (copy of API block used by generator)
*       will be updated to turn off accumulation by the main process
*       done upon CHARACTER subroutine (see above)
CHAR_CHAR_0200 DS     0S
            IF (TM,$BCB_PFLAG2,$BCB_CCSID_CNV,NZ)
*               IF IREG =2 AND SRC_ACCUM TR INST WILL BUMP REG
                SETF   SAVE_R2
                IF (CLC,$BCB_IREG,EQ,=H'2'),AND,                          X
               (TM,$BCB_PFLAG1,$BCB_TRG_ACUM+$BCB_TRG_L_ACUM,NZ)
                   RESETF SAVE_R2
                   NI     $BCB_PFLAG1,X'FF'-$BCB_SRC_ACUM
                ENDIF
                RESETF XLATE_TO_E
                IF (TM,$BCB_PFLAG2,$BCB_CCSID_CNV_ATOE,O)
                   SETF XLATE_TO_E
                ENDIF
                #BAS   14,=A(DO_XTAB_SHORT)
            ENDIF
*
CHAR_9999 DS     0S
          B      CHARACTER_END
*-------------------------------------------------------------------|
* BURST CHARACTER TO CHARACTER ILEN > OLEN                           |
* TEMPLATE CODE USED FOR NON-SPACE TRUNCATION                       |
*-------------------------------------------------------------------|
CHAR_CHAR_010 DS 0S                                                 |
CHAR_CHAR_010_OLEN_A EQU *-CHAR_CHAR_010+1   LEN OF CLC             |
CHAR_CHAR_010_SDBN_A EQU *-CHAR_CHAR_010+2   LOC OF SOURCE TO COMP  |
CHAR_CHAR_010_B255_A EQU *-CHAR_CHAR_010+4   LOC OF 255 SPACES      |
          CLC    0(0,0),0(0)                 SDB+(OLEN-1),BWK_BUFF255
CHAR_CHAR_010_BERR_A EQU *-CHAR_CHAR_010+2                          |
          BNE    0(R12)                      NOT SPACES? -> #@ERROR4
CHAR_CHAR_010_L   EQU *-CHAR_CHAR_010                               |
*-------------------------------------------------------------------|
```

Fig. 6h